INDEX

No	Topic	Page No
1	React Js Tutorial Introduction	1
2	React Features	4
3	Pros and Cons	6
4.	React Js vs Angular Js	9
5.	React Js vs React Native	10
6.	React vs Vue	12
7.	React JSX	13
8.	React Components	18
9.	React State	22
10	React Props	25
11	React Props Validation	28
12	React State vs Props	31
13	React Constructor	32

14	React Component API	35
15	React Component Life Cycle	40
16	React Forms	42
17	Controlled Vs UnControlled	46
18	Conditional Rendering	49
19	React Lists	51
20	React Keys	52
21	React Refs	53
22	React Fragments	56
23	React Router	58
24	React CSS	60
25	React Animation	66
26	React Map	68
27	React Table	70
28	Higher Order Component	71

29	React Code Spliting	73
30	React Context	75
31	React Hooks	77
32	React Flux vs MVC	78
33	React Redux	81
34	React Portal	82

ReactJs

ReactJs is a declarative, efficient, and flexible Javascript library for building reusable UI components. It is an open-source, component-based front end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook

Why we use ReactJs :-

The main objective of ReactJs is to develop User Interface (UI) that improves the speed of the apps. It uses virtual DOM (Javascript Object), which improves the performance of the app. The javascript virtual DOM is faster than the regular DOM. We can use ReactJs on the client and server-side as well as with other frameworks. It uses component and data patterns that improves readability and maintain larger apps.

- **React create-react-app**

The create-react-app is an excellent tool for beginners, which allows you to create and run React project very quickly. It does not take any configuration manually. This tool is wrapping all of the required dependencies like Webpack, Babel for React project itself and then you need to focus on writing code.

Requirement :-

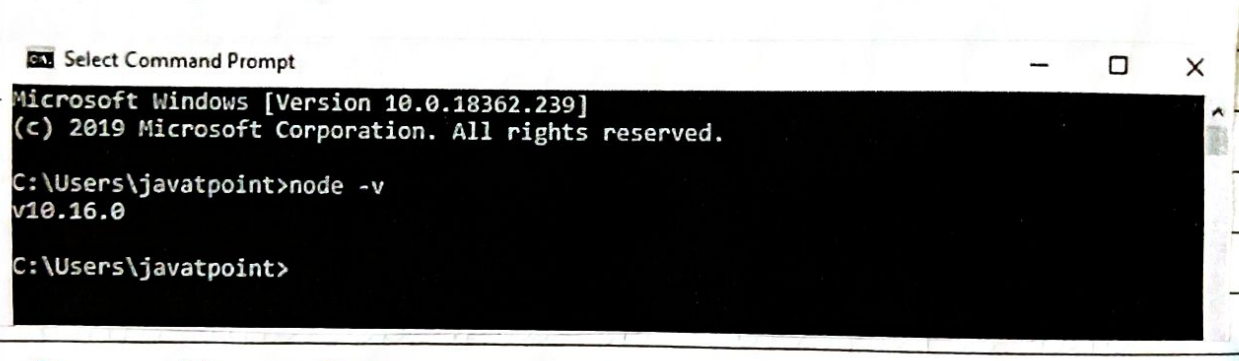
The create React App is maintained by Facebook and can works on any platform for ex- macOS, Windows, Linux.

To create a React project using create react-app you need to have installed the following things.

1] Node version >= 8.10
2] NPM version >= 5.6

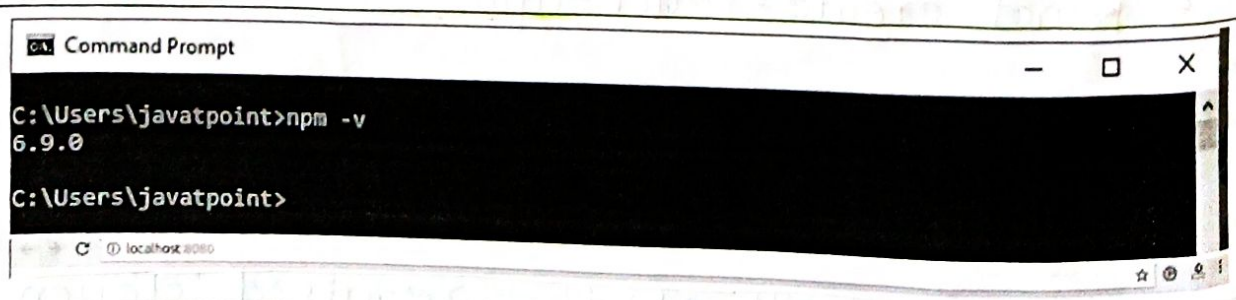
- Run the following code to check the Node version in command prompt.

```
$ node -v
```



- Run the following command to check the NPM version in command prompt.

```
$ npm -v
```



In react application, there are several files and folders in the root directory, some of them are as follows.

1] node-module :- It contains the React library and any other third party libraries needed

2] public :- It holds the public assets of the application. It contains the index.html where React will mount the application by default on the   element.

3] Src :- It contains the App.css, App.js, index.js and serviceWorker.js, Here the App.js file always responsible for displaying the output screen react.

4] package-lock.json :- It is generated automatically for any operations where npm package modifiers either the node-module tree or package.json. It cannot be published. It will be ignored if it finds any other place rather than the top-level package.

5] package.json :- It holds various metadata required for the project. It gives information to npm, which allows to identify the project as well as handle projects.

6] Readme.md :- It provides the documentation to read about React topics

React Features :-

JSX :-

JSX stands for Javascript XML. It is a java-Script syntax extension. Its an XML or HTML like syntax used by ReactJs. This syntax is proceed into javascript calls of React Framework. It extends the ES6 so that HTML like text can co-exit with Javascript react code. It is not necessary to use JSX, but it is recommended to use in ReactJs.

Component :-

ReactJs is all about components, ReactJs application is made up of multiple compone and each component has its own logic and control These components can be reusable which helps you to maintain the code when working on larger scale projec

One-way Data Binding :-

ReactJs is designed in such manner that follows unidirectional data flow or one-way data binding. The benifits of one-way data binding give you better co throught application. If the data flow is in another direction then it require a additional feature. It is because compon are supposed to be immutable and the data within them cannot be changed.

- **Virtual DOM :-**

A virtual Dom object is a representation of the original DOM object. It works like a one-way data binding. Whenever any modifications happen in the web application. The entire UI is re-rendered in virtual DOM representation and new DOM. Ones it has done, the real Dom will update only the things that have actually changed.

- **Simplicity :-**

ReactJs uses JSx file which makes the application simple and to code as well as understand. We know that ReactJs is component based approch which makes the code reusable as your need. This makes it simple to use and learn.

- **Performance :-**

ReactJs is known to be a great performer. This feature makes it much better than other framework out there today. The reason behind this is that it manages virtual Dom The DOM is a cross platform and programming API which deals with HTML, XML & XHTML. The DOM exits entirely in memory Due to this when we create a component we did not write directly to DOM.

Pros and Cons ReactJs :-

Advantage of ReactJs :-

① Easy to learn and use :-

→ ReactJs is mush easier to learn and use It comes with a good supply of documentat tutorial and training resource.

② Creating Dynamic Web-Application Becomes Easier :-

To create a dynamic web application specifically with HTML Strings was tricy because it requires a complex coading, but ReactJs solved that issue and makes it easier. It provide less coding and gives more functionality. It makes use of JSX, which is particular syntax leting HTML Quotes.

③ Reusable Component :-

A ReactJs web application is made up a multiple components. and each components has its own logic and controls. These components are responsible for outputing small and reusable code helps to make your apps easier to develop and maintain

④ Performance Enhancement :-

ReactJs improves performance due to virtu Dom. The Dom is a cross platform and programming API which deals with HTML XML or XHTML. Most of the developers faced the problem when the Dom was

updated. which slowed down the performance of the application. ReactJs solved this problem by introducing virtual Dom.

⑤ The Support of Handy Tools :-
ReactJs has also gained popularity due to the presence of handy set of tools. These tools make the task of developer understandable and easier. The React Developer Tool have been designed as Chrome and Firebox. dev extension and allow you to inspect the React component hierarchies in the virtual Dom.

⑥ Known to be SEO Friendly :-
Traditional Javascript framework have an issue in dealing with SEO. The search engins generally having trouble in reading JS application. Many web developers have often complained about this problem. ReactJs overcomes this problem that helps developers to be easily navigated on various search engins.

⑦ The Benifits of having Javascript Library :-
ReactJs is choosing by most of the web developers. It is because it is offering very rich Javascript library. The javascript library provides more flexibility to the web developers to choose the way they want.

Disadvantage of ReactJS

① The high pace of development:-
The high pace of development has an advanta and disadvantage both. In case of disadvantage since the Envirnment continuesly change so fast, some of the developers not filling comfertable to relearn the new ways of doing things regullarly.

② Poor documentation:-
It is another cons which are common for constanly updating technology. React technology and accelerating so fast that there is no documentation properly.

③ View Part:-
React Js cover only the UI layers of the app nothing else, so you still need to choose some other technologies to get complete tooling set for development in project

④ JSX is barrier:-
React Js uses JSX. Its syntax extension that allows HTML with Javascript mixed together. This approch has its own benefits, but some members of the development community consider JSX as a barrier especially for new developers. Developers complain about its complexity in the learning curve.

ReactJs vs AngularJs :-

	AngularJs	ReactJs
Author	Google	Facebook community
Developer	Misko Hevery	Jordan Walke
Initial Release	october 2010	March 2013
Latest Version	Angular 1.7.8 on 11 March 2019	React 16.8.6 on 27 March 2019
Language	Javascript, HTML	JSX
Type	Open source MVC Framework	Open Source JS Framework.
Rendering	Client-Side	Server-Side
Packaging	Weak	Strong
Data-Binding	Bi-directional	Unidirectional
Testing	Unit and Integration Testing	Unit Testing
App Architecture	MVC	Flux

Dependencies	It manages dependencies automatically	It require additional tools to manage dependencies
Routing	It requires a template or controller to its router configuration, which has to be manged manually.	It doesn't handle routing but has a lot of module for routing.
Performance	Slow	Fast, due to virtual Dom.
Best For	It is best for single page application that update a single view at a time	It is best for single page applications that update multiple views at a time

React Js Vs React Native (Difference)

	React Js	React Native
①	The ReactJs initial release was in 2013	The react Native initial release was in 2015
②	It is used for develop	It is used for developing

	web application	mobile applications
③	It can be executed on all platform	It is not platform independant. It takes more effort to be executed on all platform
④	It uses a Javascript library and CSS for animations	It comes with built-in animation libraries.
⑤	It uses React-router for navigating web pages	It has built-in animation libraries.
⑥	It uses HTML tags	It does not use HTML tags.
⑦	It can HTML tags	It does not use HTML tags
⑧	It can use code components, which saves a lot of valuable time	It can reuse React Native UI component & modules which allow hybrid apps to render natively.
⑨	It provides high security	It provides low security in comparison to ReactJs

React Vs Vue

	React	Vue.
Defination	React is declarative efficient, flexible open-source JS library for building reusable UI Component	Vue is an open-source JS library for building reusable user interfaces and single page application
History	It was created by Jordan Walke, a software engineer at Facebook, It was initially developed and maintained by Facebook and later used in its products like Whatsapp & Insta.	Vue was created by Even You, a former employee of Google worked on many Angular project. He wanted to better version of Angular, just extracting the part which he liked about Angular and making it lighter.
Prefered language	Javascript / XML	HTML / Javascript
Size	The size of the React library is 100 kilobytes	The size of the Vue library is 60 kb.

Performance	The performance is slow as compared to Vue	Its performance is fast as a compared to React.
Flexibility	React provides great flexibility to support third party library	Vue provides limited flexibility as compared to React.
Current Version	React 16.8.6 on March 27, 2019	Vue 2.6.10 on March 20, 2019
Long Term Support	It is suitable for long term support.	It is not suitable for long term support.

React JSX.

JSX provides you to write HTML/XML like structure in the same file where you write javascript code, then preprocessor will transform these expression into actual Javascript code. Just like XML/HTML, JSX tags name, attributes, and chilldren.

Why use JSX?

- It is faster then regular javascript because it performs optimization while translating the code to javascript.
- Insted of separating technologies by putting markup and logic in separate files, React

uses component that contain both. We will learn component

- It is type-safe, and most of the errors can be found at compilation time.

- It makes easier to create template.

Nested Elements in JSX :-

App.JSX

```
import React {component} from 'react';
class App extends Component {
  render () {
    return (
      <div>
        <h1> Javatpoint </h1>
        <h2> Training Institute </h2>
        <p> This website contains
        the best CS tutorials </p>
        </div>
    );
  }
}
export default App;
```

JSX Attribute :-

JSX use attributes with the HTML elements same as regular HTML. JSX uses convention for attribute rather the standard naming convention of HTML

Such as a class in HTML becomes class-Name in JSX because the class is the reversed keyword in javascript. We can also user our own custom attribute, we need to use data-prefix.

Example:-

```
import React,{Component } from 'react';
class App extends Component {
  render() {
      return (
          <div>
            <h1> JavaTPoint </h1>
            <h2> Training Institute </h2>
          <P data -demoAttribute ="demo">
    This website contains the best CS</P>
    </div>
  );
  }
}
export default App;
```

In JSX, We can specify attribute values in 2 ways;

① As String Literal:-

We can specify the values of attributes in double quotes.

```
var element = <h2 className ="firstAttribute">
  Hellow JavaTpoint </h2>
```

Example:-

```
import React, { Component } from 'react';
class App extends Component {
    render () {
        return () {
            <div>
        <h1 ⇒ class Name = "hello" > JavaTpoint</
        <p data-demo Attribute = " demo" >
        This website contains the best cs
        tutorials </p>
    </div>
  );
  }
}
export default App;
```

2] As Expressions:-

We can specify the values of attributes as expression using curly braces { }

Example.

```
import React, { Component } from 'react';
class App extends Component {
    render () {
        return (
            <div>
            <h1 class Name = "hello" > (25+20)
                                        </h1>
        </div>
    );
  }
}
export default App;
```

JSX Comments

JSX allows us to comments that begins with /* and ends with */ and wrapping them in curly braces { } just like in the case of JSX expressions

Example

```
import React, {Component} from 'react';
class App extends Component {
    render (){
        return (
            <div>
          <h1 className="hello" > Hello JavaPoint</h
          </div>
        );
    }
}
export default App;
```

JSX Styling

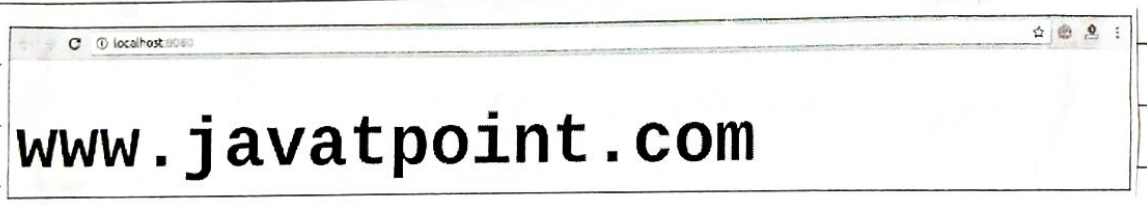
React allways recommended to use inline style To set inline styles, you need to use camel-case syntax. React automatically allows appending px after the number value on specific elements

Example :-

```
import React, {Component} from 'react';
class App extends Component {
 render (){
   Var myStyle = {
       fontSize : 80
```

```
font-family: 'Courier',
 Color : '#003300'
}
return (
    <div>
      <h1 Style = {mystyle} >
            www.javatpoint.com </h1>
      </div>
  );
 }
}
export default App;
```

Output:-



React Components

React Component

Functional Component | Class Component

Functional Component:-

In React, function component are a way to write component that only conte

a render method and don't have their own state. They are simply JavaScript function that may or may not receive data as a parameter. We can create function that takes props as input and returns what should we rendered.

```
function WelcomeMessage (props) {
  return <h1> Welcome to the ; {Props.name}
                                  </h1>
  }
```

The functional component is also known as a stateless component because they do not hold or manage state.

Example.

```
import React, {Component} from 'react';
class App extends React.Component {
  render () {
    return(
      <div>
        <First/>
          <Second/>
          </div>
    );
  }
}
class First extends React.Component {
    render () {
      return (
      <div>
        <h1> JavaTPoint </h1>
      </div>
  );
```

```
  };
}
Class Second extends React.Component {
    render (
        <div>
        <h2> www.JavaTpoint </h2>
        <p> This website contains the
                great CS tuto </p>
    </div>
  );
 }
}
}
export default App;
```

Class Components.

Class components are more complex than functional components. It requires you to extend from React. Component and create a render function which returns a React element. You can pass data from one class to other class component.

Example:-

```
import React {Component} from 'react';
class App extends React.Component {
constructor () {
```

```
Super ();
this.state = {
  data:
    [  {
          "name":"Abhishek"
       }
       {  "name" : "Saharsh"
       }
       {
          "name": "Ajay"
       }
    ]
  }
}   render() {
      return (
          <div>
  <h1> Student Name Detail </h1>
  </div>
  );
} }
Class List extends React.Component {
  render () {
   return (
      <ul>
        <li> {this.props.data.name} </li>
      </ul>
  };
} }
export default App;
```

React State :-

The state is an updated structure that is use to contain data or information about the component. The state in component can change over time. The state in change over time happen as a response to user action or system event.

A state must be kept as simple as possible It can be state by using the setState() method and calling setState() method trigger UI Updates. A state represents the component's local state or information. To state an initial state before any interaction occure we need to use getInitialstate.. method.

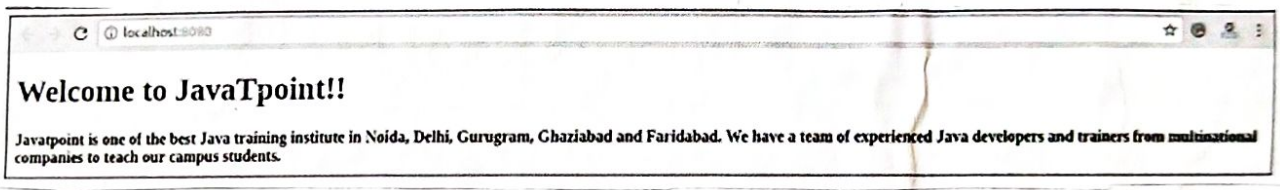
Defining State:-

To define a state, you have to first declare a default set of values for the defining the components initial state.

Example:-

```
import React, {componet} from 'react';
class App extends React Component {
  Constructor (){
    Super ();
    this. state = {display Bio : true};
  }
  render (){
    const bio = this.state.displayBio ?(
    <div>
<h3> Javatpoint is one of the best training
```

```
institute In Noida ,Delhi, Gurugram, Ghaziabad
</h3> </p>
    </div>
    ): null ;
     return (
    <div>
        <h1> Welcome to javaTpoint </h1>
        {bio}
    </div>
    );
  } }
export default App ;
```



Changing the state :-

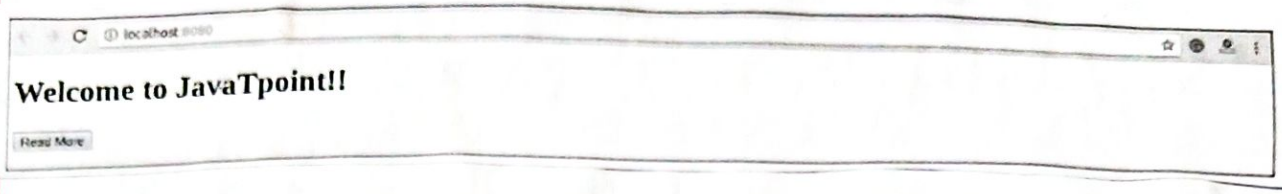
We can change the component state by using setstate() method and passing a new state object as the argument Create a new method toggle-DisplayBio() in the above example.

```
this.toggleDisplay Bio = this.toggleDisplayBio.bind
                                        (this);
```

```
import React {Component} from 'react';
class App extends React.Component {
  Constructor () {
  Super ();
  this.state ={display Bio : false};
```

```
console.log ('component this', this );
 this.toggleDisplayBio = this .toggleDisplayBio
                                bind (this);
   }
this.toggleDisplayBio() {
      this. setState ({display Bio :! this. state
                        displayBio });
   }
  render () {
     return (
        <div>
  <h1> Welcome to Javatpoint </h1>
   {
       this.state. displayBio? (
          <div>
<h4> JavaTpoint is one of the best Java
  institute in Noida</h4> </p>
     <button onclick ={ this.toggleDispla
        Bio } > Show Less </button>
     </div>
    ); (
        <div>
<button onclick = {this.toggleDisplayBio}>
     Read more </button>
    </div>
  ) }
   </div>
 ) } } }
export default App;
```

when you click on Read More



React Props:-

Props stand for Properties. They are read-only components. It is an object which store the valu of attribute of a tag and work simillar to the HTML attribute.

Props are immutable so we cannot modify this props from inside the component.

Example:-

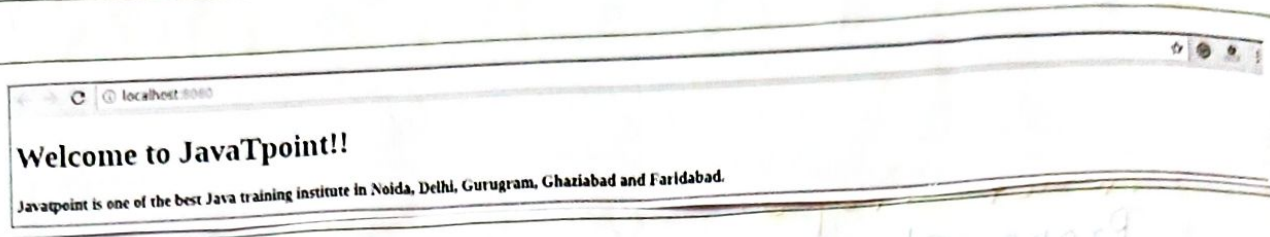
App.Js

```
import React ,{ componeht } from 'react';
class App extends React.Component {
  render ( ){
    return (
      <div>
        <h1> Welcome to {this.props.name} </h
      <P> <h4> Javatpoint is one of the best
        training institute in India </h4> </di
    );
  } }
export default App;
```

Main.Js

```
import React from 'react'
import ReactDom from 'react-dom';
import App from '/App.Js';
```

```
React Dom. render (<App name ='Java Tpoint" />,
document. get Element By Id ('app'));
```



Default Props :-

It is not necessary to always add props in the ReactDom.render () element. You can also set defaults props directly

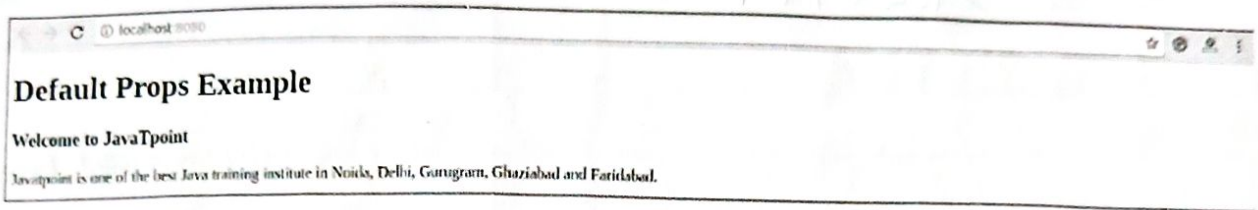
Example

App.Js

```
import React , {Component } from 'react';
 class App extends React.Component {
  render () {
     return (
      <div>
      <h1> default Props Example </h1>
      <h3> Welecome to {this.props. name </h
      <P> JavaTpoint is one of the best Java
      training  </P>
      </div>
   );  } }
  App. defaultProps = {
      name : "JavaTpoint"
   }
  export default App;
```

Main.Js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from '/App.js';
ReactDOM.render (<App/>, document.getElement-
ById('app'));
```



State and Props.

It is possible to combine both state and props in your app. You can set the state in the parent Component and pass it in the child component using props.

Example.

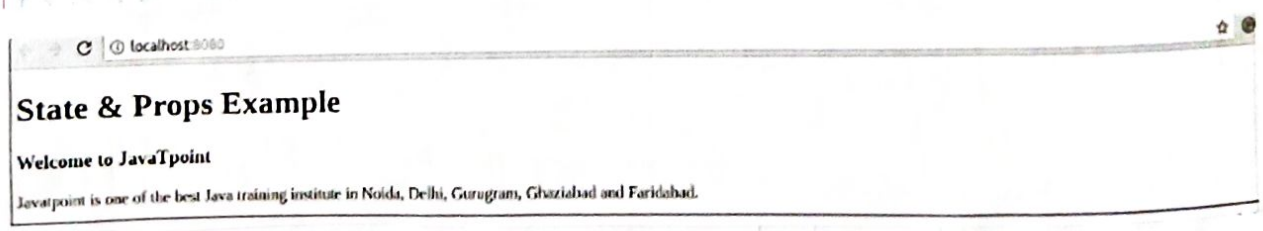
App.Js

```
Import React {Component} from 'react';
class App extends React.Component {
  Constructor (props) {
    Super (props);
      this.state = {
        name : "JavaTpoint"
      }
    }
  render () {
    return {
  <div>
    <JTP  jtpProps = {this.state.name}/>
```

```
  </div>
 ); } }
class JTP extends React.Component {
 render () {
   return (
    <div>
    <h1> State & Prope example </h1>
    <h3> Welcome to {this.props.jtpProp} </h3>
    <P> JavaTpoint is one of the best Java
      training institute in Noida </P>
  </div>)
 ); } }
export default App;
```

Main.Js

```
import React from 'react';
import ReactDom from 'react-dom';
import App from '/App.Js';
ReactDom.render (<App/>, document.get-
  ElementById ('app'));
```



React Props Validation:-

Props are an important mechanism for passing the read-only attribute to React component. The props are usually requir to use correctly in the component.

Validating Props :-

App.propsType is used for props validation in react component. When some of the props are passed with an invalid type, you will get the warnings on Javascript console. After specifying the validation patterns.

Syntax:-

```
class App extends React.Component {
    render() {}
}
Component.propTypes = {/*    */}
```

ReactJs Props Validator.

	Props Type	Description.
①	PropType.any	The props can be of any data type.
②	PropTypes.array	The props should be an array.
③	PropTypes.bool	The props should be a boolean.
④	PropTypes.func	The props should be a fun
⑤	PropTypes.number	The props should be a number
⑥	PropTypes.Object.	The props should be a object

7	PropTypes.string	The props should be String
8	PropTypes.Symbol	The props should be Symbol
9	PropTypes.instanceOf	The props should be an instance of particular Javascript class.
10	PropTypes.isRequired	The props must be provided
11	PropTypes.element	The props must be element
12	PropTypes.node	The props can render anything : number, string, element or an array containging these types.
13	PropTypes.oneOf()	The props should be one of several types of specific values.
14	PropTypes.oneOfType ([PropTypes.string, Prop.Types.number])	The props should be an object that could be one of many type

- **React Js Custom Validators.**

ReactJs allows creating a custom validation function to perform custom validation. The following arguments is used to create a custom validation function.

Props :- It should be first argument in the Component.

propName :- It is the propName that going to validate.

ComponentName :- It is the componentName that are going to validate again.

Example:-

```
var Component = React.createClass({
App.propTypes = {
 customProp : function (props, propName, compo-
   nentName) {
 if(!item.isValid (props [propName])) {
    return new Error ('Validation failed');
  } } } } )
```

Difference between State and Props.

Props	State.
Props are read-only	State changes can be asynchronous.
Props are immutable	State is mutable
Props allow you to pass data from one component to other component as an argument.	State holds information about the components
Props can be accessed	State cannot be accessed

by the child component	by child components
Props are used to communicate between component	State can be used fo rendering dynamic ch- anges with the componen
Stateless component can have Props	Stateless component cannot have State.
Props make component reusable	State cannot make components reusable
Props are external and controlled by whatever render the component	The State is internal and controlled by the React Component itself.

SN	State and Props
①	Both are plain JS Object
②	Both can contain default values.
③	Both are read-only when they are using by this.

What is Constructor:-

The constructor is a method used to initiallize an object State in class. It automatically called during the creation of an object in

class.

```
Constructor (props) {
    super (props);
  }
```

In react, Constructor are mainly used.

① It is used for initializing the local state of the component by assigning an object in this. state.

② It is used for binding event handler methods that occure in your component.

Example :-

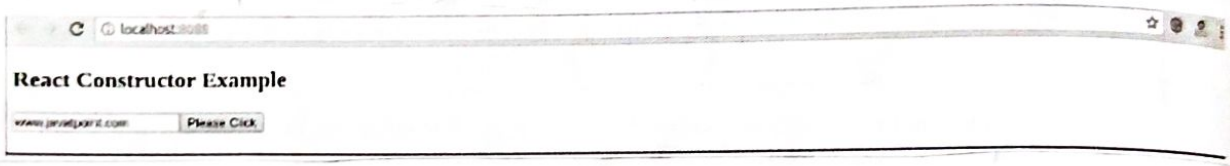
App.JS

```
import React, { Component } from 'react';
class App extends Component {
  constructor (props)
   Super( props);
   this.state = {
     data : " www. javatpoint. com")
      }
   this.handleEvent = this.handleEvent.bind (this);
  }  handleEvent () {
       Console.log ( this. props);
   } render( ) {
  return (
    <div ClassName = "App" >
    <h2> React Constructor </h2>
  <input type = "text"  Value = {this. state. data}
                                                  />
<button onClick = {this.handleEvent } >
        Please Click </button>
```

```
</div>
); }}
export default App;
```

Main-Js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from '/App.Js';
ReactDOM.render (<App />. document .get-
    ElementById ('app'));
```



Arrow Function:-

The arrow function is the new feature ES6 standard. If you need to use arrow functions. It is not necessary to bind any event to 'this'.

```
import React, {Component} from 'react';
class App extends Component {
  Constructor ( props ) {
      Super (props);
    this.state = {
        data : 'www.javatpoint.com'
    }}
  handleEvent = ( ) => {
  Console.log (this.props );
  }
```

```
render (){
 return (
    <div ClassName ='App'>
  <h2> React Constructor </h2>
   <input type = "text " value ={this. state. data}/>
   <button onClick ={this.handleEvent} >
                Please Click </ button>
    </div>
  ); }}
 export default App;
```

React Component API.

React Js Component is a top-level API. It makes the code completly individual and reusable in the application. It includes various method

- creating element.
- Transforming element.
- Fragments

① setState ()

This method is used to update state of the component. This method does not always replace the state immediatly. It only adds change to the original state. It is primary method that is used to update the user interface (UI) in response (UI) in event handler and server response.

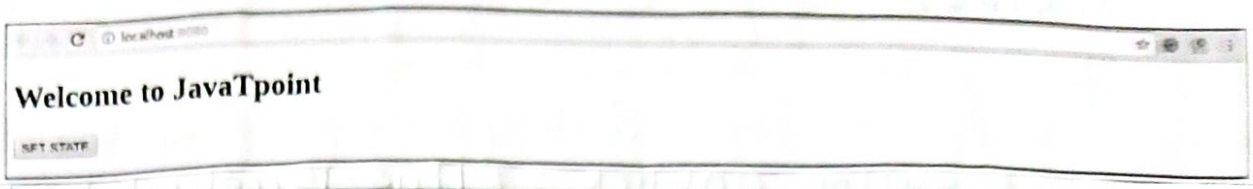
```
this.state.State (object newState [, function
   callback ]);
```

```
import React, {Component} from 'react';
import PropTypes from 'prop-types';
Class App extends React.Component {
  Constructor () {
   Super ();
  this.state ={
    msg : "Welcome to JavaTpoint"
  };
  this.updateSetState = this.updateSetState
          .bind (this);
  }
  updateSetState () {
      this.setState ({
       msg."Its best tutorial".
      } );   }
  render () {
      return (
        <div>
      <h1> {this.state.msg} </h1>
     <button onClick ={this.updateState }>
        SetSTATE </button>
     </div>
  ); }}
  Export default App;
```

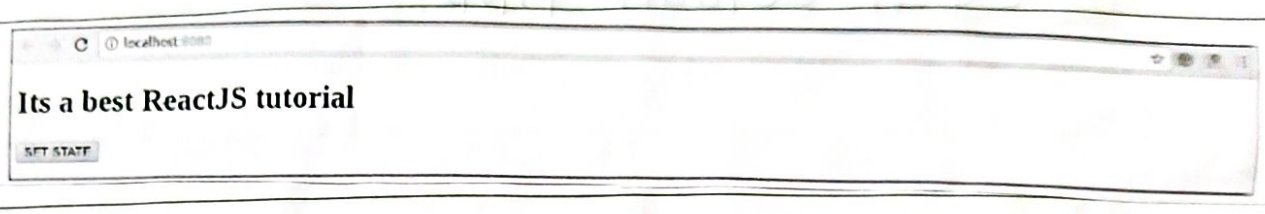
Main.Js

```
import React from 'react';
Import ReactDOM 'react-dom';
Import App from '/App.Js';

ReactDOM.render (<App/>, document.getElem
ntById ('app'));
```



When you click on SET STATE button, you will see following screen with updated message.



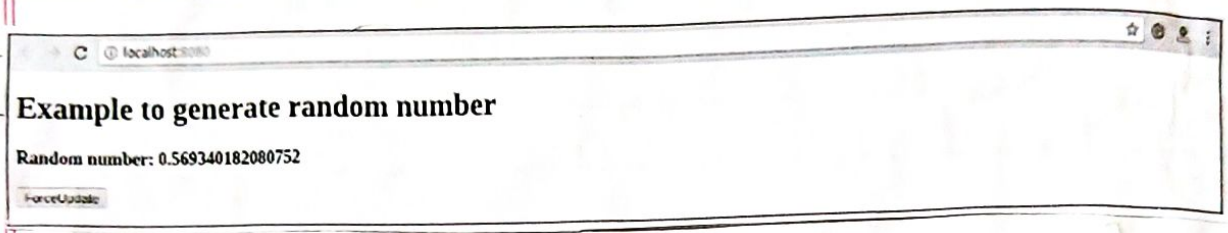
forceUpdate()
This method allows us to update the component manually.

```
Component.forceUpdate(callback);
```

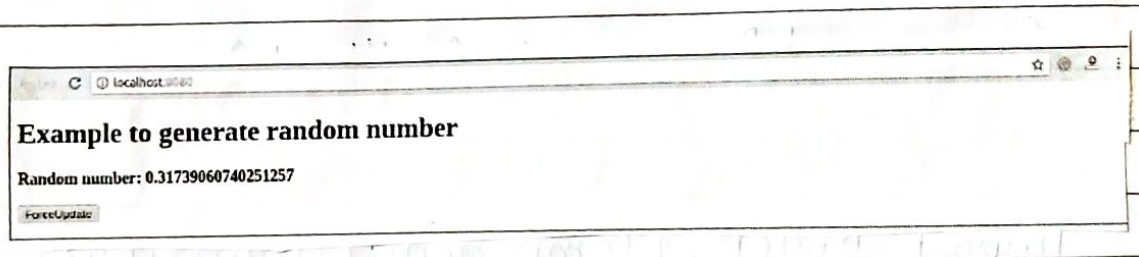
Example. App.Js.

```
import React, {Component} from 'react';
class App extends React.Component {
  constructor(){
    Super();
    this.forceUpdateState = this forceUpdateState.
    bind(this);
  }
  forceUpdateState(){
        this.forceUpdate();
  };
render() {
    return (
```

```
<div>
<h1> Example to generate random Number</
 <h3> Random number ; { Math.random() } </h
 <button onClick = {this.forceUpdateState} >
        ForceUpdate </button>
 </div>
); }}
export default App;
```



Each time when you click on ForceUpdate button.



find DOMNode()

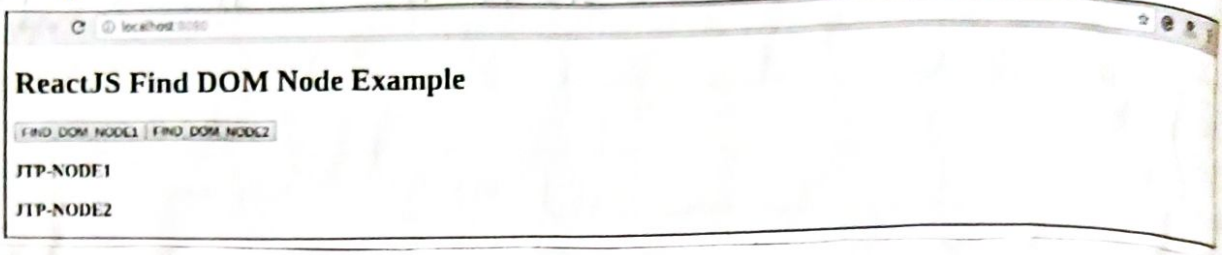
For DOM manipulation, you need to use ReactDom.findDOMNode() method. This met allows us to find or access the underl DOM node.

```
ReactDOM.findDOMNode(component);
```

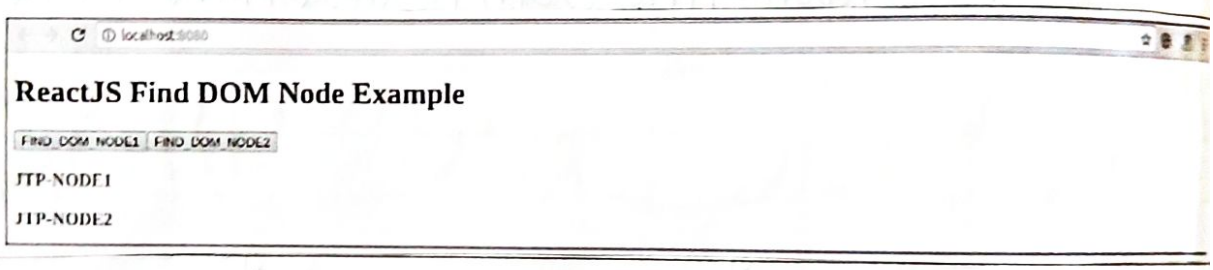
App.Js

```
import React {Component} from 'react';
import ReactDom from 'react-dom';
class App extends React.Component{
 constructor (){
  Super ();
 this.findDOMNodeHndler1 = this.findDomNode-
      Handler1.bind (this);
 this.findDOMNodeHandler2 = this.findDOMNode-
      Handler2.bind (this);
 };
 findDomNodeHandler1() {
    var myDiv =document.getElementById ("
        myDivOne');
  ReactDom.findDOMNode (myDivOne).style.
     color='red';
 }
findDomNodeHandler2 (){
    var myDiv =document.getElementBy Id
                ('myDivTwo');
   ReactDom.findDomNode (myDivTwo).style
      .color ='blue';
 }
render (){
    return (
    <div>
   <h1> ReactJs Find DOM Node Example </h1
  <button onClick ={this.findDomNodeHandler1}
     FIND-DOM-NODE 1 </button>
  <button onClick ={this.findDomNodeHandler2}
     FIND-DOM- NODE 2 </button>
    <h3 id = "myDivOne"> JTP-Node1 </h3>
```

```
<button onClick = {this.forceUpdateState}
    ForceUpdate </button>
  </div>
  );
} }
export default App;
```



Ones you click on the button, the color of the node gets changed.



React Component Life-Cycle.

In ReactJs, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component lifecycle. These lifecycle methods are not very complicated and called at various during a componants life.

Initial Phase → Mounting phase → Updating phase → Unmounting phase.

① Initial Phase -

• getDefaultProps ()

It is used to specify the default value of this.props. It is invoked before the creation of the component or any props from the parent is passed into it.

• getInitialPhase()

It is used to specify the default value of this.state. It is invoked before the creation of the component.

② Mounting Phase

• ComponentWillMount ()

This is invoked immediately before a component gets renderded into the Dom. In the case when you call setState () inside this method

• ComponentDidMount ()

This is invoked immediatly after component gets rendered and placed on the Dom. You can do any Dom querying operations.

• render()

This method is defined in each and every component. It is responsibble returning a single root HTML node element.

③ Updating Phase.

• ComponentWillRecieveProps ()

It is invoked when a component receives new props. If you want to update the

state in response to prop changes, you should compare this.Props and nextprops to perform state transition by using this.setState()

- ShouldComponentUpdate()
It is invoked when a component decides updating occures.

- Component Will Update()
It is invoked just before the component updating occurs. Here you can't change the component state by invoking this.setState() method.

- Component Did Update()
It is invoked immediatly after the component updating occurs. In this method, you can put any course inside this.

4] Unmounting Phase
Component Will Unmount()
This method is invoked immediatly before a component is destroyed permanantly. It perform necessary cleanup.

React Forms:-

Form are an internal part of any modern web application. It allows the users to interact with the applications as well as gather information from the user. Forms can perform many tasks that depends on the nature of your buisness requirements and

logic such as authentication of the user.

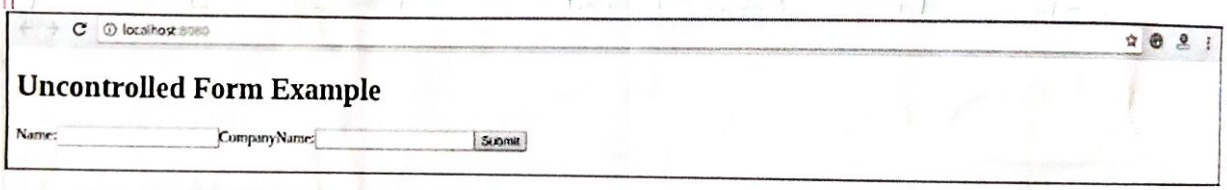
Creating Form :-
React offers a stateful, reactive approch to build a form. The component rather than Dom usually handles the React form.

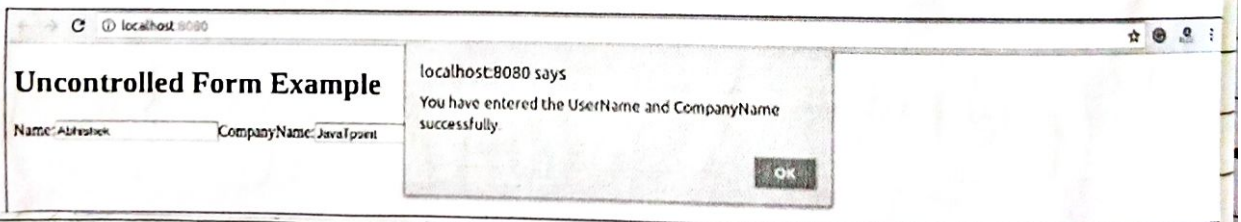
① Uncontrolled component.
② Controlled component.

① Uncontrolled Component :-
The uncontrolled component input is simillar to the traditional HTML form input. The Dom itself handles the form data. The HTML element maintain their own state that will be updated when the input values changes. To write an uncontrolled component.

```
import React, {Component} from 'react';
class App extends React.Component {
    constructor (props)
        super(props);
        this.updateSubmit = this.updateSubmit.
                                bind (this);
        this.input = React.createRef ();
    }
updateSubmit (event) {
 alert ("You have entered the username and
  CompanyName successfully");
  event.preventDefault();
 }
render () {
```

```
return()
  <form onSubmit={this.updateSubmit}>
    <h1> Uncontrolled Form </h1>
    <label> Name:
      <input type = "text" ref = {this.input
    </label>
    <label>
    Company Name:
    <input type = "text" ref={this.input1/>
    </label>
    </form>
  );
}}
export default App;
```





Controlled Component:

Controlled component have functions that gov ern the data passing into them on every onChangeevent; rather than grabbing the data only ones.

```
import React {Component } from 'react';
class App extends React.Component {
    constructor (Props) {
      super (props);
     this.state = {value :" "};
     this.handleChange = this.handleChange.bind
                                        (this);
    this.handleSubmit= this.handleSubmit.bind(this);
  }
   handleChange(event) {
     this.setState({value: event.target.value})
   }
   handleSubmit (event) {
     alert ("You have submitted the input Success-
          fully:' + this.state.value);
      event.preventDefault ();
   }
  render ( ){
    return(
    <form onSubmit = {this.handleSubmit} >
     <h1> Controlled Form </h1>
     <label>
     Name:
   <input type ="text" value = {this.state.value}
        onChange = {this.handlechange} />
     </label>
     <input type = "submit" value ="Submit "/>
    </form>
    );
  }
}
export Default App;
```

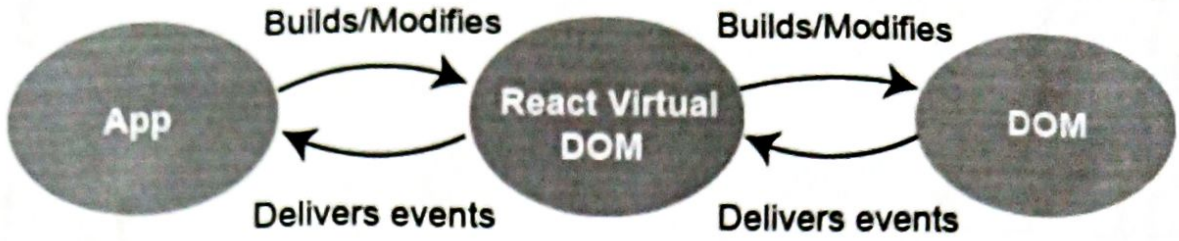
Controlled Componant Vs Uncontrolled Component.

Controlled	Uncontrolled.
It does not maintain internal state	It maintains internal state.
Data is controlled by parent component	Data is controlled by a Dom itself
It accept the current value as a prop	It uses a ref for their current values.
It allows validation control	It does not allow validation control.
It has better control over the form elements and data	It has limited control over the form and elements.

React Events :-

React has its own event handling system which is very simillar to handling events on Dom elements. The react event handling system is known as Synthetic Events. The synthetic events is a a cross browser wrapper of the browser native event.

① React events are named as camelcase insted of lowercase.



② A function is passed as a event handler insted of a string.

For declaration in plain HTML:-

```
<button onclick="showMessage()">
    Hellow JavaTpoint
</button>
```

Event declaration in React:-

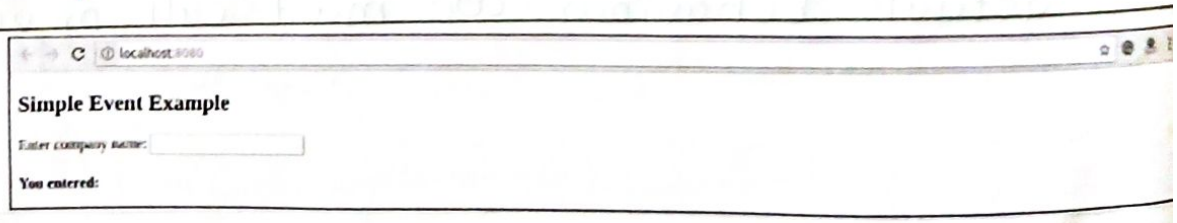
```
<button onClick = {showMessage}>
    Hellow JavaTpoint
</button>
```

③ In react, we cannot return false to prevent the default behaviour we must call preventDefault event explicitly to prevent the

Example:-

```
import React, {Component} from 'react'
class App extends React Component{
    constructor (props)
        Super (props)
        this.state = {
            CompanyName = ''
        };
```

```
};  }
 Change Text (event) {
     this.setState({
       company Name: event.target.value
     });
 }
 render ( ){
   return ( ){
       <div>
     <h2> Simple Event </h2>
    <label html For='name"> Enter Company
                Name </label>
   <input type="text" id="company Name"
       onchange = {this.change Text bind (this
     <h4> You entered ; {this.state.company
     Name } </h4>
    </div>
  );
  }
}
export default App;
```



React Conditional Rendering:-

There is more than one way to do conditional rendering in React.

- if
- It is the easiest way to have conditional rendering in React in the render method.

```
function UserLogin (props){
    return <h1> Welcome Back </h1>;
}
function GuestLogin(Props ) {
    return <h1> Please Sign Up </h1>
}
function SignUp (props){
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn ) {
      return <User Login />;
  }
  return <GuestLogin />;
}
ReactDom.render {
  <signUp isLoggedIn = {false} />
  document.getElementById ('root')
);
```

Logical && operator.

The operator is used to checking the condition.

```
import React from 'react';
import ReactDOM from 'react-Dom';
function Example ()
```

```
return (<div>
  { (10>5) && alert ('This alert will be
          Shown!")
  }
  </div>
);
}
```

Ternary Operator :-

```
render ( ) {
  const isLoggedIn = this.state.isLoggedIn;
  return {
   <div>
  Welcome {isLoggedIn ? 'Back ; Please login}
  </div>
  };
}
```

Switch Case Operator

```
functional NotificationMsg ({text}) {
  Switch (text) {
   case 'Hi all';
        return <Message: text ={text }/>;
   case "Hellow JavaTpoint";
        return <Message text ={text}/>;
   default
    return null;
  }
}
```

Conditional Rendering with enums

An enum is a great way to have a multiple Conditional rendering. It is more readable as compared to switch case operator.

```
function NotificationMsg ({text ,state}) {
  return (
    <div>
   {{
     info: < Message text = {text} />
          warning : < Message text = {text} />
     } [state]}
   </div>
   );
  }
```

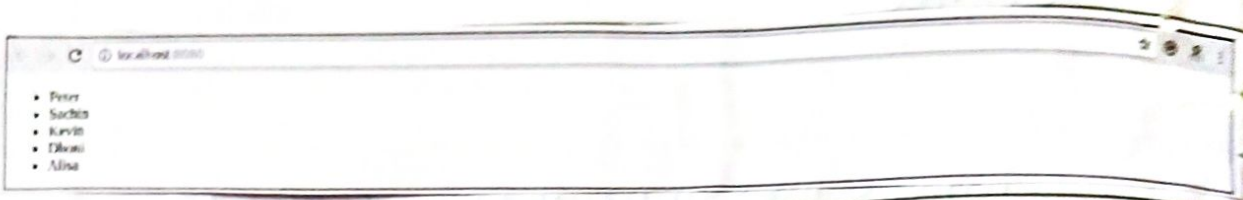
React Lists:-

Lists are used to display in an ordered forma and mainly used to display menus on websit The map() function is used to traversing the list. Include the new list <ul> elements and render it to dom.

```
import React from 'react';
import ReactDOM from 'react-dom';
const myList = ['Peter', 'sachin', 'kelvin', 'Dhoni
  'Alisa'];
Const myListItems = myList.map ((myList) => {
    return <li> {mylist } </li>
  });
ReactDOM.render (
   <ul> {listItems} </ul>
```

```
document.getElement By Id ('app')
);
export default App;
```

Output.



React Keys :-

A key is unique identifier. In React, It is used to identify which items have changed, updated or deleted from the lists. It is useful when we dynamically created component or when the user alert the Lists. It also helps to determine which component in a collection need to be rendered insted of re-rendering the entire set of components every time.

```
const StringList = ['Peter', 'Sachin, 'Kevin',
        'Dhoni', 'Alisa'];
const updatedLists = StringLists.map ((StrList
    <li key ={strList.id} > {StrList} </li>
  } );
```

If there are no stable IDs for rendered items, you can assign the item index as a key of the lists. It can be shown in the below,

```
const stringLists = ['Peter', 'Suchin', 'Kevin', 'Dhoni',
        'Alisa'];
const updatedLists = stringLists.map ((strList, index)=>
    {
    <li key={Index} > {strList} </li>
  });
```

React Refs:-

Refs is the shorthand used for referances in Reac
It is an attribute which makes it possible to store a referance to particular DOM nodes or React elements. It provides a way to access React DOM nodes or React elements and how to interac with it.

When to Use Refs:-

- When we need DOM measurements such as managing focus, text selection, or media playback.
- It is used triggring imperative animations.
- When integrating with third-party DOM libraries.
- It can also use as in callbacks.

When to not use Refs:-

- It is use should be avoided for identify anything that can be done declaratively. insted of using open() and close() methods on a Dialog component, you need to pass an isOpen prop to it.
- You should have to avoid overuse of the Refs.

How to create Refs.

In React, Refs can be created by using React.createRef(). It can be assigned to React element via the ref attribute. It is commonly assigned to an instance property when a component is created and them can be referenced throughout the component.

```
class MyComponent extends React.Component {
  constructor (props) {
  super (props);
  this.callRef = React.createRef( );
  }
  render ( ) {
    return <div ref={this.callRef} />
  } }
```

How to access Ref's

When a ref is passed to an element inside render method. a referance to the node can be accessed via the current attibute of the ref.

```
const node = this.callRef.current;
```

Ref current Properties

- When the ref attribute is used in HTML element, the ref created with React.createRef() receives, the underlying DOM element as its current property.

- If the ref attribute is used on a custom class component then ref object receives the mounted instance of the component as its current property.

- The ref attribute cannot be used to on function component because they dont have instances.

Callback Refs :-

There is another way to use refs that is called 'callback refs' and it gives more control when the refs are set and unset. Insted of creating refs by passing a callback function to the ref attribute of a component.

```
<input type ="text" ref ={element => this.callRefInput =element}/ >
```

The callback function is used to store a referance to the DOM node in an instance property and can be accessed elsewhere.

```
this.callRefInput.value
```

React with useRef ()

It is introduced in React 16.7 and above version. It helps to get access the DOM node element and the we can interact with the DOM node or element such as focusing the element or accessing the input element value. It returns the ref object whose current property initiallized to the passed argument.

```
const refContainer = useRef(initialValue);

function useRefExample (){
    const inputRef = useRef (null);
    const onButtonClick =()=>{
    inputRef.current.focus();
  };
  return (
    <>
  <input ref={inputRef} type="text"/>
  <button onClick={onButtonClick}> Submit
                                  </button>

    </>
  );
}
```

React Fragments:-

The render method can return single elements or multiple element. The render method will only render a single node inside at a time.

```
class App extends React.Component {
    render (){
      return (
    <div>
    <h2> Hellow World </h2>
    <P> Welcome to the JavaTpoint </P>
    </div>
    );
  }
}
```

Syntax :-

```
<React Fragment>
  <h2> Child 1 </h2>
  <p> child 2 </p>
  ........
</React. Fragment>
```

Why we use Fragments?

① It makes the execution of code faster as compared to the div tag.

② It take less Memory.

Fragments Short Syntax:-

There is also another shorthand exists for declaring fragments for the above method. It looks like empty tag in which we can use '<>' and insted of the 'React.Fragment'

```
Class Colums extend React.Component {
 render ( ){
    return (
    <>
   <h2> Hello World! </h2>
    <P> Welcome to the JavaTPoint </P>
    </>
    );
  }
}
```

Key Fragments:-

The shorthand syntax does not accept key

key attribute. You need a key for mappi
a collection to an array of fragments su
as to create a description list.

```
Function = (props) {
  return (
    <Fragment>
  {props.item.data.map(item => (
    <React.Fragment key = {item.id}>
      <h2> {item.name} </h2>
      <p> {item.url} </p>
      <p> {item.description} </p>
    </React.Fragment>
  ))}
  </Fragment>
  )
}
```

React Router :-

Routing is a process in which a user is dire
to diffrent pages based on their action
or request. ReactJs Router mainly use
for developing Single Page application.
React Router is used to define multi
routes in the application.
React Route is a standard library syst
built on the Top of the React and use
to create routing in the React applic
using React Router package. It provid
the synchronous URL on the browser
with data that will be displayed o
the web page.

React contain 3 diffrent package of routing
① react-router :- It provides the core routing Componants and function for the React Router application.

② react-router-native:-
It is used for mobile application.

③ react-router-dom :-
It is used for web application design.

$npm install react-router-dom --save

Components in React Router.
<Browser Router>
- It is used for handling URL

<HashRouter>
- It is used to handling static request.

Benefits of React Router:-
① It is not necessary to set the browser history manually.

② Link uses a navigate the internal links in the application. It is simillar to anchor tag.

③ It uses switch feature for rendering

④ If the router needs only a single child element.

React CSS

CSS in React is used to style the React App or component. The style attribute is the most attribute for styling in React applications. which adds dynamically-computed styles at render time. It accepts a javascript object in camelcased properties rather than CSS string

(1) Inline Styling

The inline styles are specified with a javascript object in camelcase version of the style name. Its value is the styles value which we usually take in a string.

App.Js

```
import React from 'react';
Import ReactDom from 'react-dom';
class App extends React Component{
  render() {
    return (
  <div>
  <h1 style={{ color:"Green"}} Hello JavaT</h1>
  <P> Here you can find all CS Tutorials </P>
  </div>
  );
  }
}
export default App;
```

② CSS Stylesheet :-

You can write styling in a seperate file for your React application, and save the file with a CSS extension. You can import the file in your application.

App.Js.

```
import React from 'react';
import ReactDom from 'react-dom';
import '/App.CSS';
class App extends ReactComponent {
  render (
   <div>
   <h1> Hello JavaTpoint </h1>
   <p> You can find all cs tutorials </p>
  </div>
  );
} }
export default App;
```

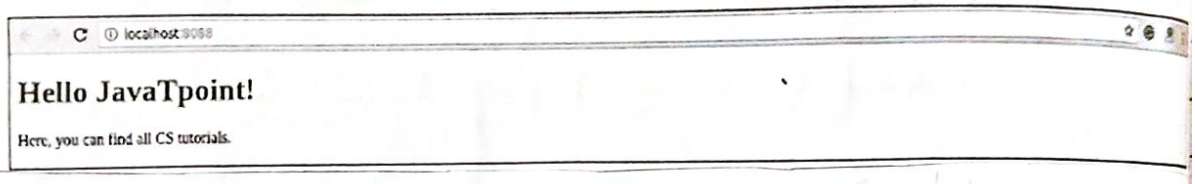
App.CSS

```
body {
 background color : #008080;
  color : Yellow;
  padding : 40 px;
   font-family : Arial;
    text-align : center;
  }
```

Index.html

```
<!DOCTYPE html>
  <html lang = "en" >
```

```
<head>
<meta charset ="utf-8" />
 <meta name = "viewport"
  content = "width-device-width,
            initial-scale =1" />
 <title> React App </title>
 </head>
 <body>
   <div id ="app" /> </div>
 </body>
</html>
```



③ CSS Module

CSS Module is another way of adding styles to your application. In a css file where all class names and animation names are scoped locally by default. It is available only for the component which imports means any styling you add can never be applied to other components without your permission, and you never need to worry about name conflicts.

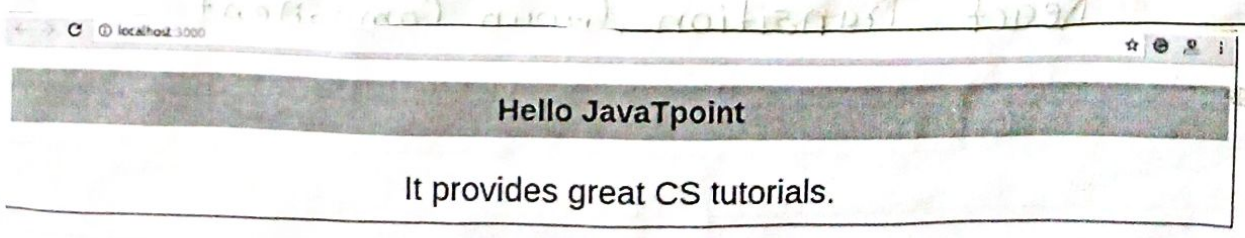
App.Js.

```
import React from 'react';
import ReactDom from 'react-dom';
import styles from '/myStyles.modules.CSS
```

```
class App extends React.component {
render() {
  return(
  <div>
  <h1 className={style.mystyle} > Hellow JavaT</h1>
  <P className={style.parastyle} It provides great
   cs tutorials </p>
  </div>
  );
   }
}
export default App;
```

mystyles.module.css

```
.mystyle {
 background-color : #cdc0b0;
  color : Red;
  padding : 10 px;
   font-family : Arial
   text-align : center
  }
 .parastyle {
    color : Green;
   font-family : Arial;
   font-size : 35 px;
   text-align : center;
  }
```



4. Styled Component:-
Styled component is a library for React. It uses enhance CSS for styling React Component System in your application. which is written with mixture of Javascript and CSS.

The styled component provides.
- Automatic Critical CSS.
- No class name bugs.
- Easier deletion of CSS.
- Simple dynamic styling.
- painless maintenance.

React Animation.
The animation is a technique in which images are manipulated to appear as moving images. It is one of the most used technique to make interactive web application

React Transition group has mainly 2 API Create Transition.

① React Transition Group:-
It uses as low-level API for animation.

② React CSS Transition Group:-
It uses as a high level API for implementing basic CSS transitions and animations.

React Transition Group Component.
Provides 3 main component.
① Transition
② CSS Transition

Transition Group.

Transition:- It has a simple component API to declare a transition from one component state to another over time. It is mainly used to animate the mounting and unmounting of a component. It can also be used fro in-place transition state as well.

① entering ② entered.
③ existing ④ exited.

CSS Transition:-

The CSS transition component uses stylesheet classes to write the transition and create animations. It is inspired by the ng-animate library. It can also inherit all the props of the transition component.

- Appear
- Enter
- Exist.

Transition Group:-

This component is used to manage a set of transition component in a list. It is a state machine that controls the mounting and un-mounting components overtime. The transition component does not define any animation directly. 'Transition Group' component can have diffrent animation with a component.

App.Js

```
import React, {Component } from 'react';
import {CSS TransitionGroup } from 'react-transiti
                                    group';
class App extends React.Component {
  Constructor (props ) {
  Super (props);
  this.State ={items ['Blockchain', 'ReactJs',
        'TypeScript', 'JavaTpoint' ]};
  this.handleAdd = this.handleAdd.bind (this);
  }
handleAdd ( ){
  Const newsItems = this.state.Items.Concat([
    prompt ('Enter Item Name')
  ]);
  this.setState ({items: newsItems });
}
handleRemove (i){
  let newItems = this.state.items.slice ( );
  newsItems.splice (i, 1);
  this.setState ({items : newsItems });
}
render ( ){
  Const items =this.state.items.map((item, i)=>(
  <div key ={item } onClick ={( )=>
                    this.handleRemove (i)}>
    {item}
  </div>
));
return (
  <div>
<h1> Animation Examples </h1>
```

```
<button onClick={this.handleAdd}> Insert them
        </button>
<CSSTransitionGroup
    transitionName="example"
    transitionEnterTimeOut={800}
    transitionLeaveTimeout={600}>
    {items}
  </CSSTransitionGroup>
 </div>
 );
}
}
export default App;
```

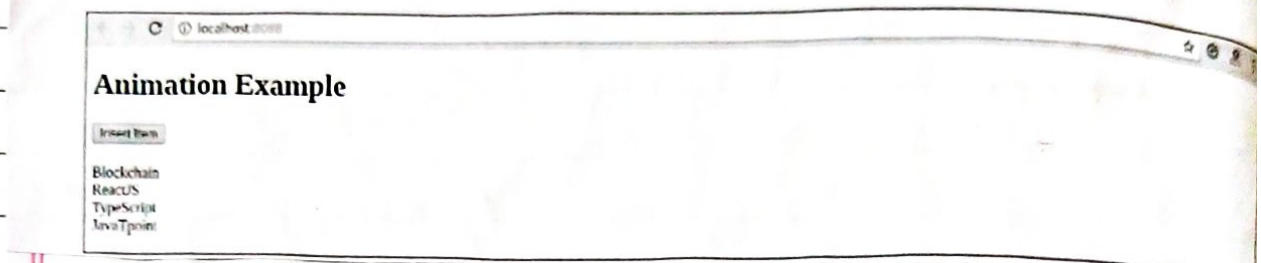
Main.Js

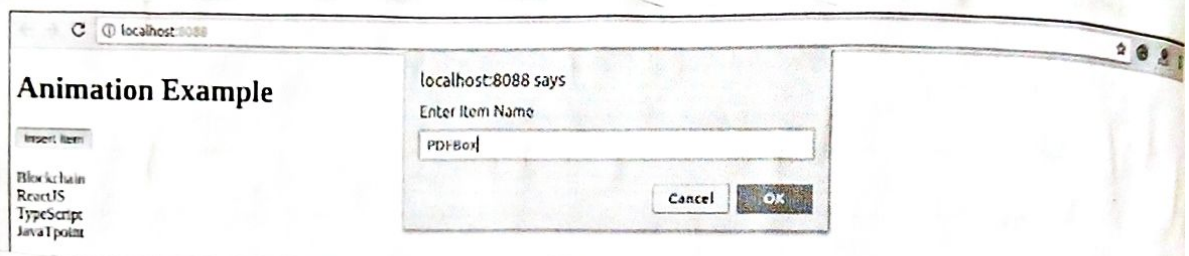
```
import React from 'react';
import ReactDOM from 'react-dom';
import App from '/App.Js';
ReactDOM.render (<App/>, document.getElementById
    ('app'));
```

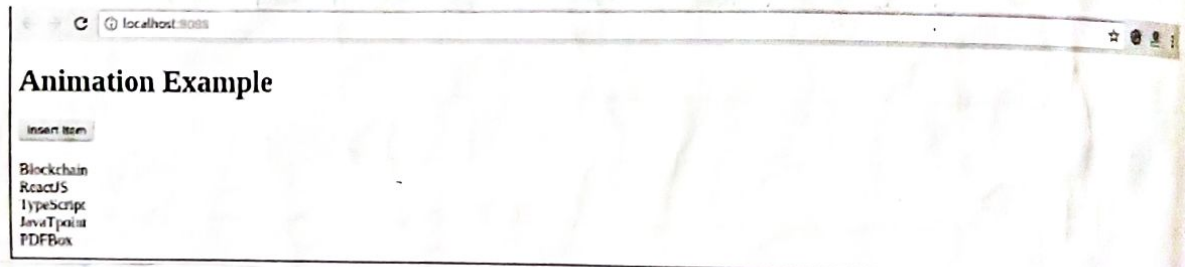
Style.Css

```
.example-enter{
  opacity:0.01;
}
.example-enter.example-enter-active{
  opacity:1;
  transition: opacity 500 ms ease-in;
}
.example-leave{
  opacity:1;
}
```

```
.example-leave.example-leave-active {
  opacity : 0.01;
  transition: opacity 300 ms ease-in;
}
```







React Map :-

A map is a data collection type where data is stored in the form of pairs. It contains a unique key. The value stored in the map must be mapped to the key. We cannot store duplicate pair in the map. It is because of the uniquness of each stored key. It is mainly used for fast searching and looking up data.

In React, the map() method used for:

① Traversing the list element.

```
import React from 'react';
import ReactDOM from 'react-dom';
function NameList (props){
  const myList(props) {
    Const listItems = myList.map((myList) =>
  <li> {myList} </li>
);
return (
  <div>
    <h2> React Map example </h2>
      <ul> {listItems} </ul>
  </div>
  );
}
const mylists = ['A','B','C','D'];
ReactDOM.render (
  <NameList mylist = {mylists} />
  document.getElementById ('app')
  );
export default App;
```

Output-

React Table :-

A table is an arrangement which organizes information into 'rows and columns. It is used to store and display data in a struct format

The react-table is light weight, fast, fully customizable and extendable Datagrid built for React. It is fully controllable via optional and callbacks.

Features:-

1. It is lightweight at 11 kb.
2. It is fully customizable.
3. It is fully controllable via optional props and callbacks.
4. It has client-side & servr-side pagination
5. It has filters
6. Pivoting & Aggregation.
7. Minimal design & easily themeble

```
import React, { Component } from 'react';
import ReactTable from "react-table";
import "react-table/react-table.css";
class App extends Component {
  render () {
    const data = [{
        name : "Ityaan";
        age : 26;
      }, {
      name : 'Ahana',
        age : 22
      }, {
```

```
name: 'Peter',
age: 40
},{
name: 'Dhoni',
age: 37
}]
Const columns = [{
Header: 'Name',
accessor: 'name'
},{
Header: 'Age',
accessor: 'age'
}]
return (
<div>
<ReactTable
data = {data}
columns = {columns}
defaultPageSize = {2}
pageSizeOptions = {[2,4,6]}
/>
</div>
) } }
export default App;
```

React Higher-Order Component

It also known as HOC. In React, HOC is an advanced technique for reusing component logic. It is a function that takes a component and returns a new component.

```
const NewComponent = higherOrderComponent
                     (wrappedComponent)
```

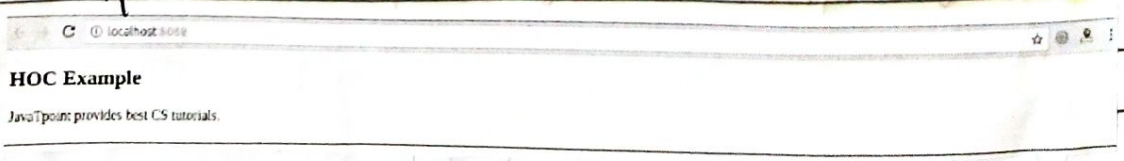
HOC.JS

```
import React, {Component} from 'react';
export default function HOC (HOCComponent){
  return class extends Component {
    render(){
      return (
      <div>
      <HOCComponent> </HOCComponent>
    </div>
  ); } }}
```

App.JS

```
import React {Component} from 'react';
import HOC from '/HOC';
Class App extends Component {
 render (
  <div>
  <h2> HOC Example </h2>
   JavaTPoint provides best CS Tutorial
   </div>
  ) } } }
  App = HOC (App);
   export default App;
```

Output -



React Code Splitting :-

The react app budled their files using tools like Webpack or Browsefy. Bundling is a process which takes multiple files and merge them into a single file, Which is called bundle. The bundle is responsible for loading an entire app at ones on the webpage.

Code splitting improves:-

1. The performance of the app.
2. The impact on memory
3. The downloadable killobytes size.

React.lazy:-

The best way for code splitting into the app is through the dynamic import () syntax. The React.lazy function allows us to render a dynamic import as a regular component.

Before:-

```
import ExampleComponent from '/ExampleComponent';
function MyComponent (){
  return (
    <div>
    <ExampleComponent />
    </div>
  );
}
```

After:-

```
const ExampleComponent = React.lazy (() =>
        import ('/ExampleComponent'));
```

```
function MyComponent () {
  return (
    <div>
   <ExampleComponent/>
   </div>
  );
}
```

Error Boundaries:-

If any module fails to load, for example, due to network failure, we will get an error. We can handle these errors with Error Boundaries. Ones we have created the Error Boundary, we can use it anywhere

```
import MyErrorBoundary from '/MyErrorBoundary'
const ExampleComponent = React.lazy(() =>
                    import ('/Example Component')
Const Example Component = React.lazy())
 Const MyComponent = () => (
   <div>
   <MyErrorBoundary>
  Suspence fallback = {<div >Loading </div>}
   <section>
    <ExampleComponent/>
    <ExampleComponent />
    </section>
    </suspense>
   </MyErrorBoundary>
  </div>
  );
```

React Context:-

Context allows passing data through the component tree without passing props down manually at every level.

React Context API :-

1. React.createContext
2. Context.provider.
3. Context-Consumer.
4. class.contextType.

① React.createContext:-

It creates a context object

```
const MyContext = React.createContext(default.value
```

② Context.Provider :-

Every Context object has a provider React component which allows consuming components to subscribe to context changes.

```
<MyContext.Provider value = { /* Some value */ />
```

③ Context.Consumer.

It is an React Component which subscribes to the context changes. It allows us to the context within an function.

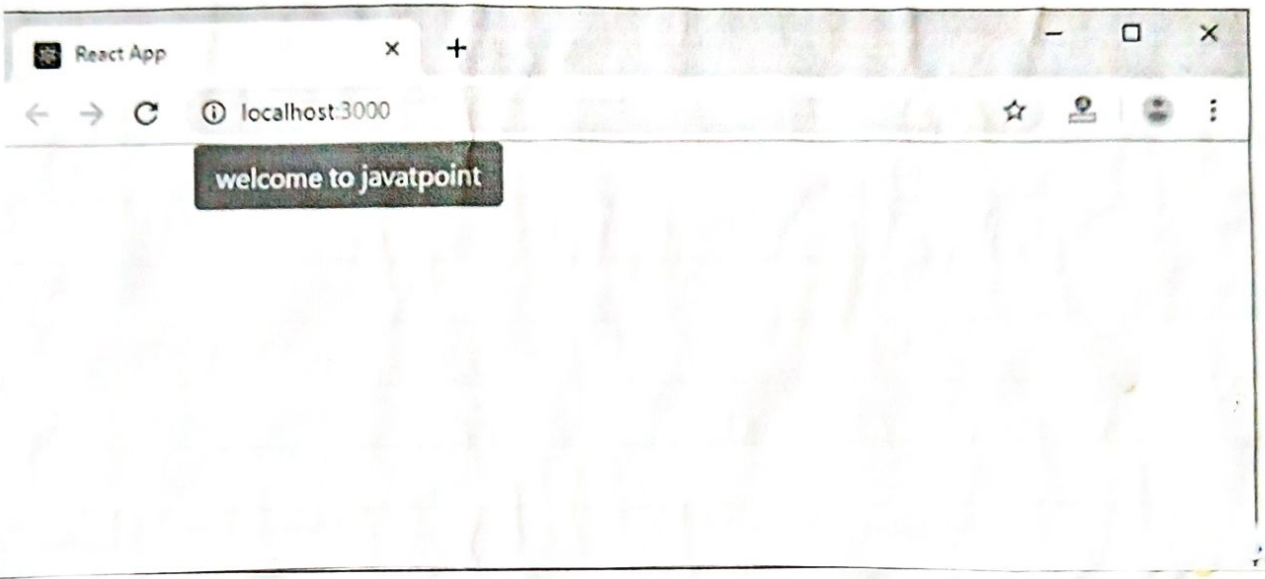
```
<MyContext.Consumer>
    {value => /* render something which is
        based on the context value */ />
```

Class contextType:-

The contextType property on a class used to asign a context object which is created by React.createContext(). It allows you to consume the closet current value of that context using this.context.

```
import React, {Component} from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
Const BtnColorContext = React.create Context ('
  btn btn-darkyellow');
class App extends Component {
  render () {
    return (
  <BtnColorContext.Provider value ='btn btn-info
  </Button>
  </BtnColorContext.Provider>
  );
 }
}
function Button (props) {
  return (
  <div className = "container">
  </ThemeButton />
 </div>
 };
}
class ThemedButton extends Component {
  static contextType = BtnColorContext;
  render () {
  return < button className = {this.context}>
  Welcome to JavaTpoint
```

```
</button>
}
}
export default App;
```



React Hooks:-

Hooks are the new feature introduced in React 16.8 version. It allows you to use state and other React feature without writing a class.

Rules of Hooks:-

1. Only call Hooks at the top level
2. Only call Hooks from React functions.

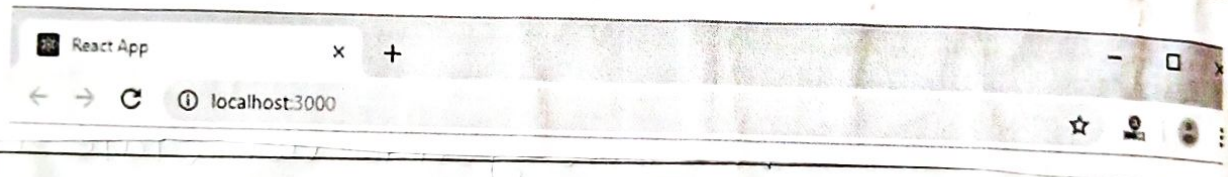
Hooks State:-

Hook state is the new way of declaring a state in React app. Hook uses useState() functional component for setting and retrivi g state.

App.Js

```
import React, {useState} from 'react';
```

```
function CountApp(){
  const [count, setCount] = useState(0);
  return (
   <div>
   <p> You clicked {count} times </p>
   <button onClick ={() => setCount(count+1
    Click me
   </button>
  </div>
  );
}
export default CountApp
```



React Flux Vs MVC

MVC

MVC stands for Model View Controller. It is an architectural pattern used for developing the user interface

MVC Architecture:-

Model:- It is responsible for maintaining th behaviour and data of an application

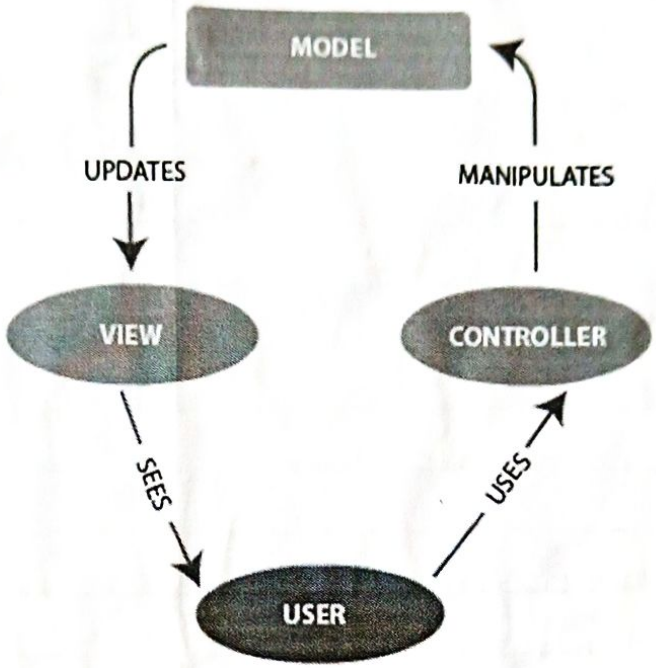
View:- It is used to display the model in the user interface.

Controller:- It acts as an interface between

the Model and the view Components. It takes user input, manipulates the data and causes view

You clicked 4 times.

Click me

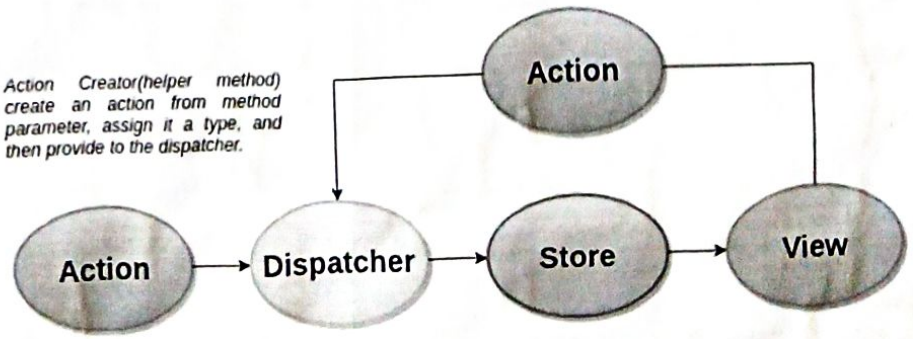


MVC Architecture

Flux:-

Flux is an application architecture that Facebook uses for building client-side web application. Flux architecture has 3 Major rules in dealing with data:-

① Dispatcher ② Store ③ views.



	MVC	Flux
1	It is introduced in 1976	It was introduced just a few years ago
2	It supports Bi-directional data Flow model	It support Uni-directional data flow model
3	In this, data binding is the key	In this events or actio are key
4	It is synchronous	It is asynchronous.
5	Controllers handle everything	Store handle all logic
6	It is hard to debug	It is easy to debug.
7	It is difficult to understand as project size increases	It is easy to the understand.
8	Its maintainability is difficult as the project scope goes huge.	It maintainability is easy and reduces run-time errors.
9	Testing of application is difficult.	Testing of application is easy.
10	Scalability is complex	It can be easily scalable.

React Redux :-

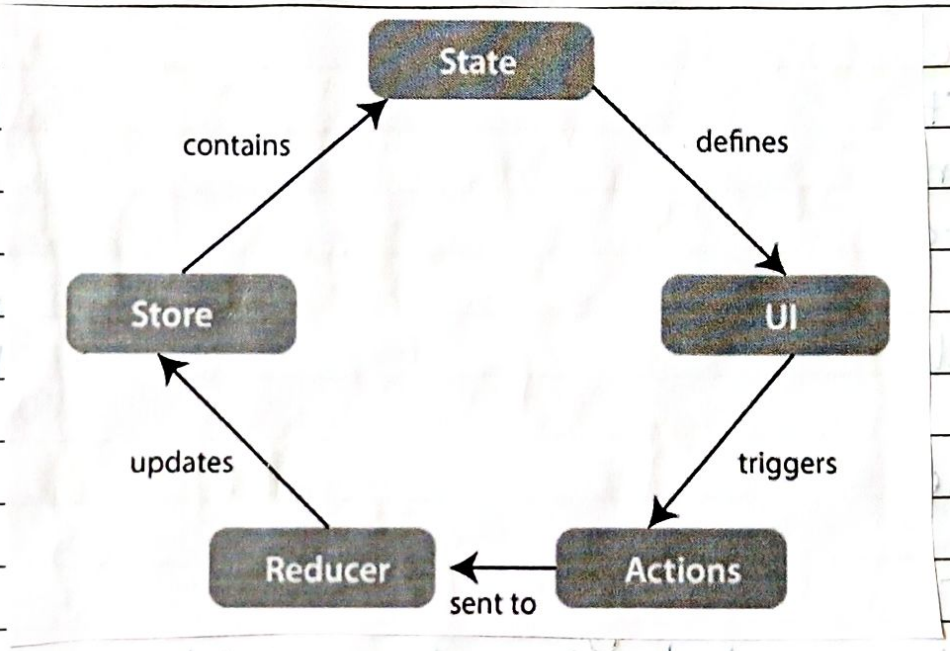
Redux was inspired by flux. Redux studied the Flux architecture.

① Redux does not have dispatcher concept.

② Redux has an only store whereas Flux has many stores.

③ The Action objects will be received and handled directly by Store.

React Architecture :-



Store :- A store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch function. It is like a brain responsible for all moving parts in Redux.

Action :- Action is sent or dispatched from the view has which are payloads that can be read by Reducers. It is pure object created

to the store the information of users event. It includes information such as type of action, time of occurances, location of occurances, its coordinates and which state it aims to change.

Reducer :- Reducer reads the payload from the action and then updates the store via the state accordingly. It is pure function to return a new state from the initial state.

React Portals :-

It is very tricky to render the child component outside of its parent component hierarchy. It breaks the convention where a component needs to render as a new element and follow a parent-child hierarchy.

ReactDOM.createPortal(child, container)

Example using Portal :-
We want to insert a child component into diffrent location in the DOM

```
render () {
 return ReactDOM.createPortal (
    this.props.chilldren,
    myNode,
 );
}
```

Features :-

- It uses React version 16 and its official API for creating portal
- It has a fallback for React version 15.
- It transports its chilldren component into a new React portal which is appended by default to document.body.
- It can also target user specified DOM element.
- It supports server-side rendering
- It supports returning arrays.
- It doesn't produce any DOM mess.
- It does no dependancies, minimalistics.

When to use ?

1. Modals.
2. Tooltips
3. Floating menus.
4. Widgets

Explanation of React Portal.

App.Js.

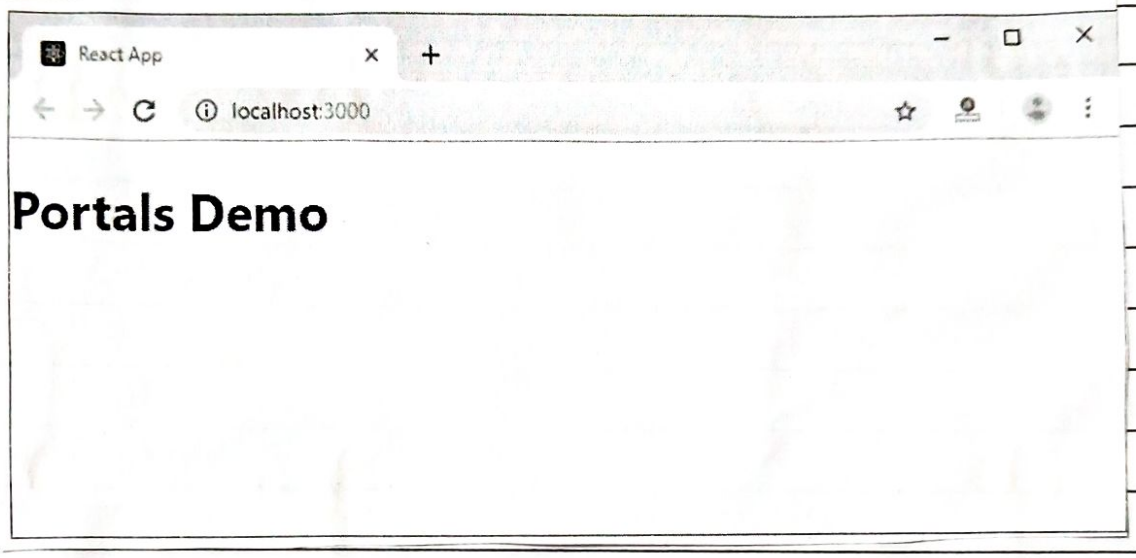
```
import React, {Component} from 'react';
import '/App.css'
import PortalDemo from '/PortalDemo.Js';
Class App extends Component {
```

```
render() {
  return (
  <div className ="App">
    <PortalDemo />
  </div>
    );
  }
}
export default App;
```

PortalDemo.Js

```
import React from 'react'
import ReactDOM from 'react-dom'
function PortalDemo() {
    return ReactDOM.createPortal(
     <h1> Portals Demo </h1>
    document.getElementById('Portal-root')
  )
}
export default PortalDemo
```

index.html.

```
<! DOCTYPE html >
 <html lang = "en">
  <head>
  <meta charset = "utf-8"/>
  <link rel ="shortcut icon" href =
                "%.PUBLIC-URL %.favicon.ico"
< meta name = "viewport" content =
        "width =device-width , initial-scale=1
<meta name ="theme-color" content =
                "#000000"/>
```

```
<title> React App </title>
 </head>
<body>
<noscript> It required to enable Javascript
    to run this app </noscript>
<div id="root"> </div>
 </body>
</html>
```